ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю»

Выполнил работу

Бондаренко Анна

Академическая группа №J3112

Принято

Практик, Дунаев Максим

Санкт-Петербург

2024

1. Введение

**Целью данной работы является разработка алгоритма поиска всех подмассивов в массиве целых чисел, сумма элементов которых равна нулю. Алгоритм должен учитывать все возможные подмассивы заданного размера K для массива из целых чисел, включающих как положительные, так и отрицательные значения, а также нули.**

**Задачи, которые решаются в рамках этой работы:**

* **Разработать алгоритм для нахождения подмассивов с суммой, равной нулю.**
* **Реализовать данный алгоритм на языке C++.**
* **Произвести анализ сложности алгоритма с учетом его времени выполнения и используемой памяти.**
* **Провести экспериментальную проверку производительности алгоритма на различных входных данных.**

1. **Теоретическая подготовка**

**Для решения задачи используется метод перебора всех возможных подмассивов заданного размера K. Подмассив — это последовательность элементов массива, расположенных подряд, длина которой равна K. Основная задача заключается в том, чтобы для каждого подмассива вычислить его сумму и проверить, равна ли она нулю. Если сумма равна нулю, то необходимо вывести индексы этих элементов.**

**Для поиска подмассивов мы будем использовать два вложенных цикла:**

1. **Первый цикл будет перебирать все возможные начальные индексы подмассивов размера K.**
2. **Второй цикл будет вычислять сумму элементов текущего подмассива и проверять условие его суммы на равенство нулю.**

**Типы данных:**

* **Основной массив хранится в виде вектора целых чисел.**
* **Индексы подмассивов также будут храниться в векторе.**

**Алгоритм имеет сложность O(N^K), где N— это размер массива, а K— размер подмассива. Это обусловлено тем, что мы перебираем все возможные подмассивы и для каждого из них вычисляем сумму.**

1. **Реализация**

**Реализация алгоритма была выполнена на языке C++. Использован стандартный контейнер vector для хранения элементов массива и индексов подмассивов. Алгоритм перебирает все возможные подмассивы размера K, начиная с каждого индекса от 0 до N−K, и проверяет сумму этих подмассивов.**

**Основные этапы реализации:**

1. Считывание входных данных: размер массива N и подмассива K, а также сами элементы массива.
2. Перебор всех подмассивов размера K.
3. Вычисление суммы каждого подмассива и проверка ее на равенство нулю.
4. Вывод индексов подмассивов, сумма которых равна нулю.
5. Экспериментальная часть

**Теоретическая оценка сложности**

Алгоритм с сложностью O(N^K), где K=8, означает, что время работы алгоритма увеличивается экспоненциально с увеличением размера входных данных. Это может привести к очень большим временным затратам при значениях N, превышающих несколько десятков. Время работы такого алгоритма не просто увеличивается линейно или квадратично, а растет как степень числа N, что требует особого внимания при его применении для больших данных.

**Подсчёт по памяти**

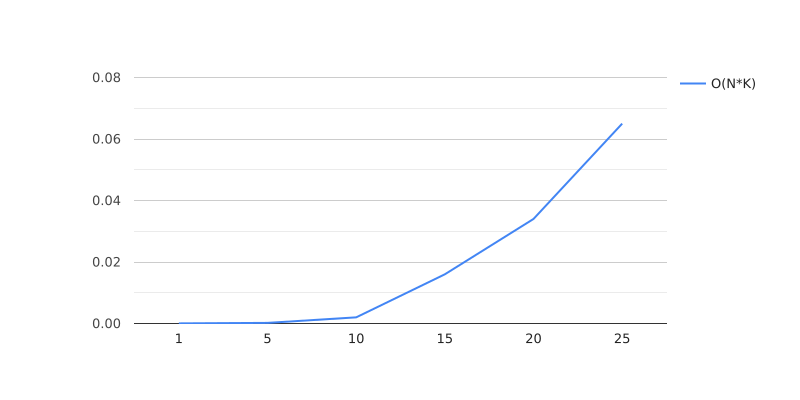
Память, используемая алгоритмом, зависит от хранения входных данных. В случае с массивом размером N, память для его хранения составляет O(N). Однако основной вклад в использование памяти не вносит сложность алгоритма, так как алгоритм использует только временные переменные для подсчета суммы и индексов. Таким образом, память остается линейной относительно размера входных данных.

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(N^K) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица 1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы, с | 0,00004 | 0,0002 | 0,002 | 0,016 | 0,034 | 0,065 |

График представляющий визуально удобный формат данных из таблицы 1 представлен на изображении 1.



Изображение №1 - График работы алгоритма

**Из таблицы и графика видно, что время выполнения программы увеличивается с увеличением размера массива N, что соответствует теоретическим ожиданиям для алгоритма с сложностью O(N^K), где K=8. При этом время выполнения растет экспоненциально с ростом N, что можно наблюдать по графику, который отображает квадратичную или даже более сложную зависимость от входных данных.**

* **При N=25 программа завершает выполнение за 0,065 секунд, что может показаться незначительным для небольших данных.**
* **Однако при большем размере N время выполнения будет стремительно расти, что делает алгоритм непригодным для обработки больших массивов данных.**

1. **Заключение**

**В ходе работы был реализован алгоритм с сложностью O(N^K), который наглядно демонстрирует экспоненциальный рост времени выполнения с увеличением размера массива N. Несмотря на то, что для малых значений N программа работает быстро, с ростом входных данных время выполнения становится неприемлемо большим.**

**Для будущих исследований можно предложить оптимизацию алгоритма с точки зрения более эффективного использования памяти, а также рассмотрение более сложных структур данных или параллельных вычислений для работы с массивами большего размера ,например, N>25.**

1. **Приложение**

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n, k;

cout << "Enter the size of the array (N): ";

cin >> n;

cout << "Enter the size of the subarray (K): ";

cin >> k;

vector<int> arr(n);

cout << "Enter the elements of the array: ";

for (int i = 0; i < n; ++i) {

cin >> arr[i];

}

for (int i = 0; i <= n - k; ++i) {

int sum = 0;

for (int j = i; j < i + k; ++j) {

sum += arr[j];

}

if (sum == 0) {

for (int j = i; j < i + k; ++j) {

cout << j << " ";

}

cout << endl;

}

}

return 0;

}